Juego de la Vida de Conway

Implementación en Scheme

Pablo Cáceres Gaitán

Programación Declarativa

Escuela Politécnica Superior de Córdoba
Universidad de Córdoba

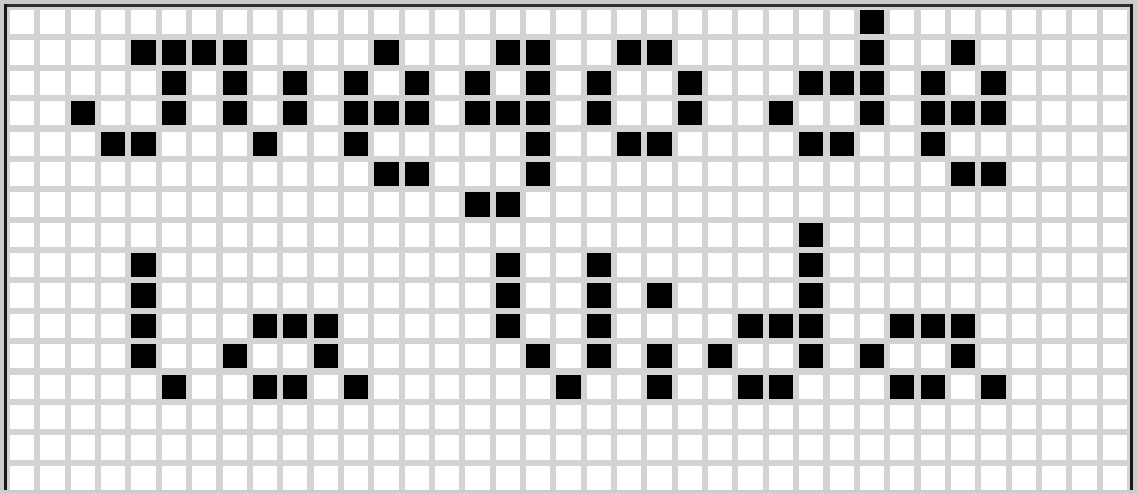
1er Cuatrimestre
Curso 2024/25

Summary

Introducción

En este trabajo trataremos la implementación del Juego de la Vida en Scheme (*Pretty Big*).

Además se creará una interfaz gráfica para interactuar con el programa.



Teoría

Teoría - Autómatas celulares

- Un **autómata celular** es un sistema dinámico discreto que evoluciona en pasos temporales.
- Se define en un retículo donde cada celda tiene un estado.
- La evolución del sistema puede depender de:
  - El estado actual de cada celda.
  - Los estados de las celdas vecinas.
  - Valores aleatorios.
- *Juego de la Vida*, *Hormiga de Langton*, *Regla 90*, ...

Teoría - El Juego de la Vida

- Diseñado por el matemático *John Conway* en 1970 [3].
- Se juega sobre una cuadrícula de celdas que pueden estar en dos estados:
  - **Viva** (negra).
  - **Muerta** (blanca).
- La evolución del juego ocurre en pasos **discretos** según **reglas** simples:
  - Una celda viva con menos de 2 vecinos vivos muere (soledad).
  - Una celda viva con 2 o 3 vecinos vivos sobrevive.
  - Una celda viva con más de 3 vecinos vivos muere (superpoblación).
  - Una celda muerta con exactamente 3 vecinos vivos nace (nacimiento).
- Este juego es un ejemplo clásico de **complejidad emergente**: reglas simples generan patrones complejos.

Teoría - Ejemplo de partida

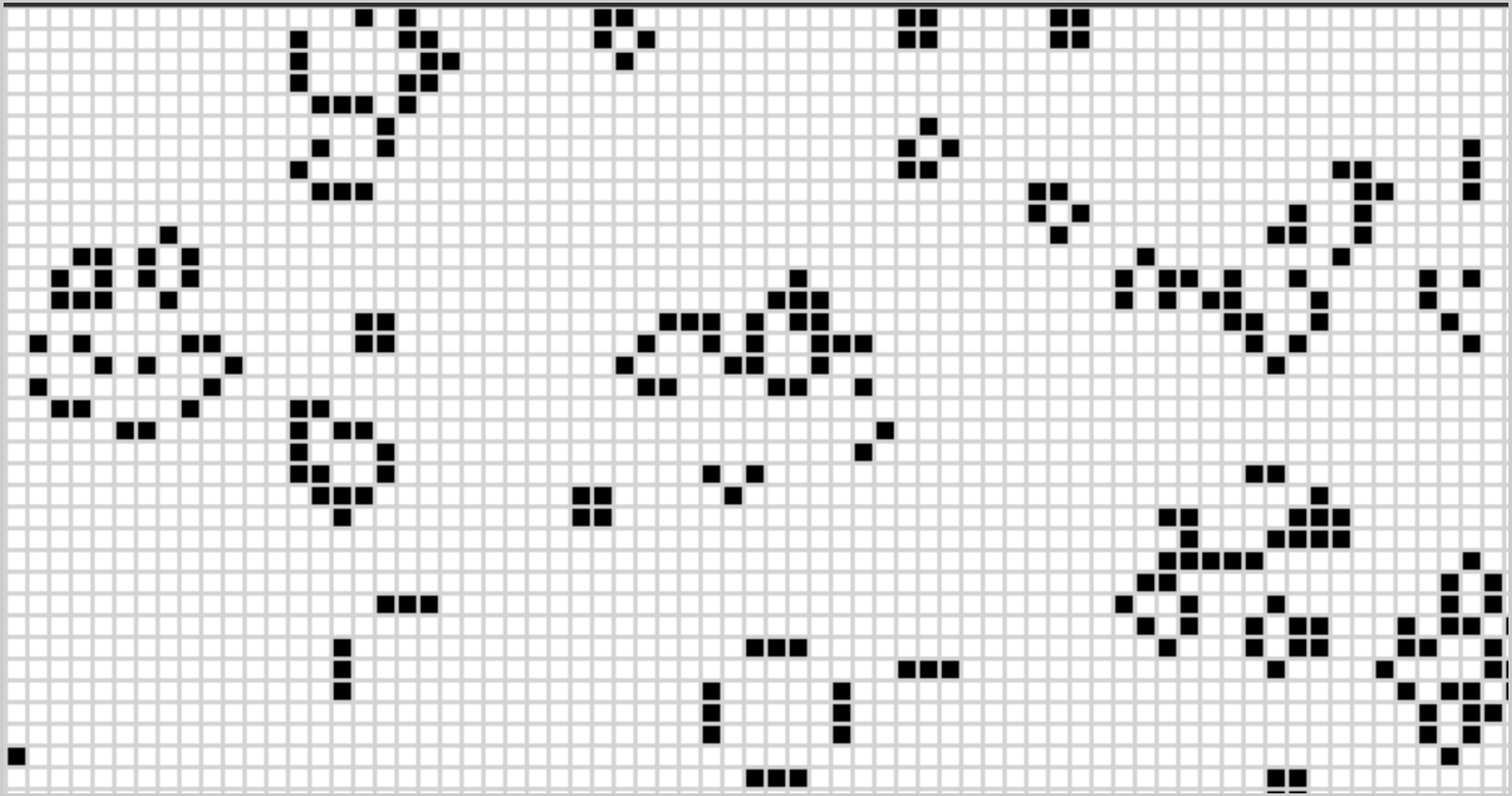


Figure: Ejemplo de tablero en una partida

Descripción Modular del Código

Descripción modular - Estructura del Proyecto

- **game_logic.rkt**: Contiene la lógica del juego.
- **game_gui.rkt**: Maneja la interfaz gráfica del usuario.
- **main.rkt**: Punto de entrada principal, lanza la aplicación.

Módulo: game_logic.rkt

La función más importante es la encargada de calcular el **próximo estado** de una **cuadrícula** en base a unas **reglas**:

```
(define (update-grid cells width height rule)
  (define neighbourhoods
    (lambda (cell)
      (get-neighbours cell width height))
    )
  (map-indexed
   (lambda (index cell)
     (rule cell (map (lambda (idx) (list-ref cells idx))
                     (neighbourhoods index))))
   cells))
```

Figure: Función *update-grid*

Módulo: game_logic.rkt

Gracias a la función anterior, definimos de forma independiente:

- **Tableros**, como listas unidimensionales.
- **Reglas**, como funciones que retornan un valor booleano.

```
(define (life-rule cell neighbours)
  (let (; Variables de let
        (alive-neighbours (count (lambda (n) (= n 1)) neighbours))
        )
    ;Cuerpo de let
    (cond
      ((and (= cell 1)
            (>= alive-neighbours min-survive)
            (<= alive-neighbours max-survive)) 1)
      ((and (= cell 0)
            (= alive-neighbours birth-neighbours)) 1)
      (else 0))
    ))
```

Figure: Función *life-rule*

Módulo: game_logic.rkt

La interfaz utiliza el módulo **racket/gui** [4]. Implementándose 3 ventanas:

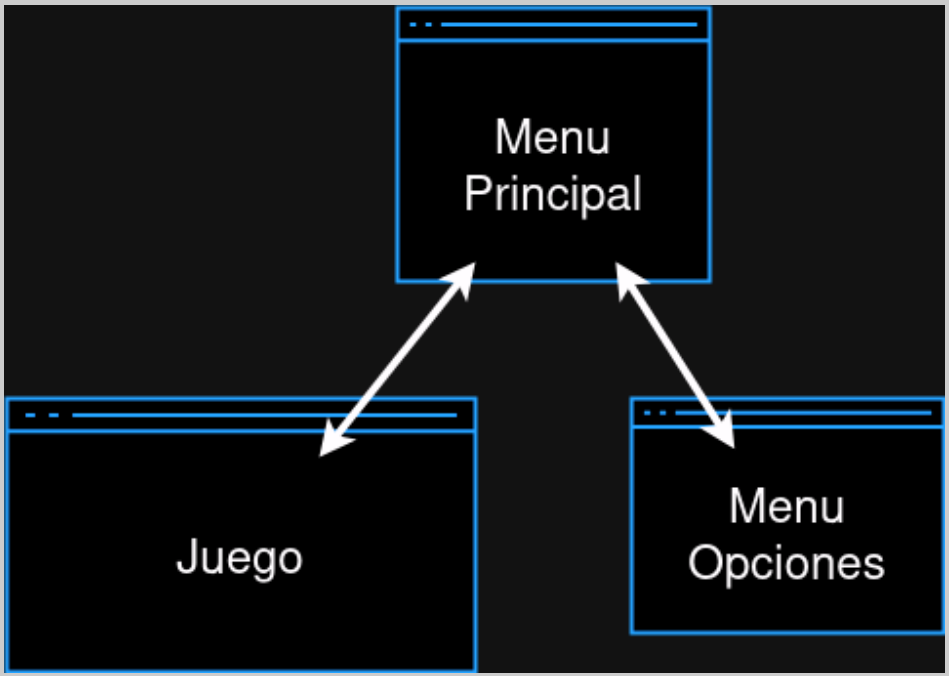


Figure: Diagrama de ventanas

Módulo: game_logic.rkt - Funcionamiento

Las ventanas se componen de un ***frame***, que contiene al resto de elementos. En el caso de los botones, deben situarse dentro de un ***panel***.

Figure: Uso de los componentes de la interfaz

Módulo: game_logic.rkt - Menú principal

El menú principal se compone de una ventana con múltiples botones para acceder a las diferentes funcionalidades del programa:

Figure: Menú principal

Módulo: game_logic.rkt - Menú de Ajustes

En el menú de ajustes se implementan diversos campos de texto para modificar los valores del juego:

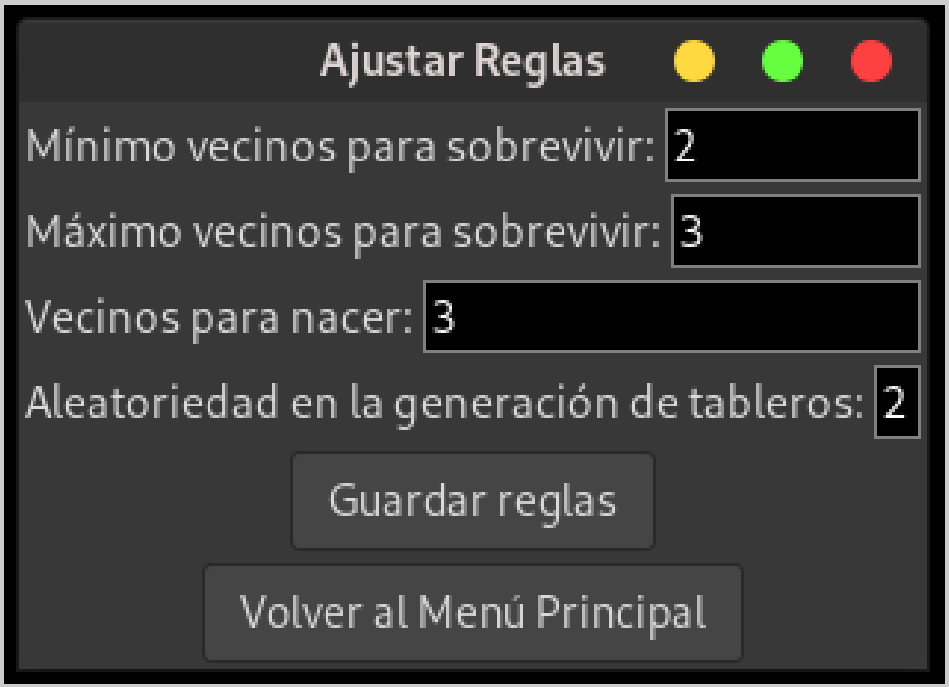


Figure: Menú de ajustes

Módulo: game_logic.rkt - Pantalla del juego

La pantalla de juego es más compleja ya que:

- Debe mostrar el tablero.
- Permite la modificación en tiempo real del estado de las celdas.
- Implementa el temporizador (variable) que hace avanzar los estados del juego.

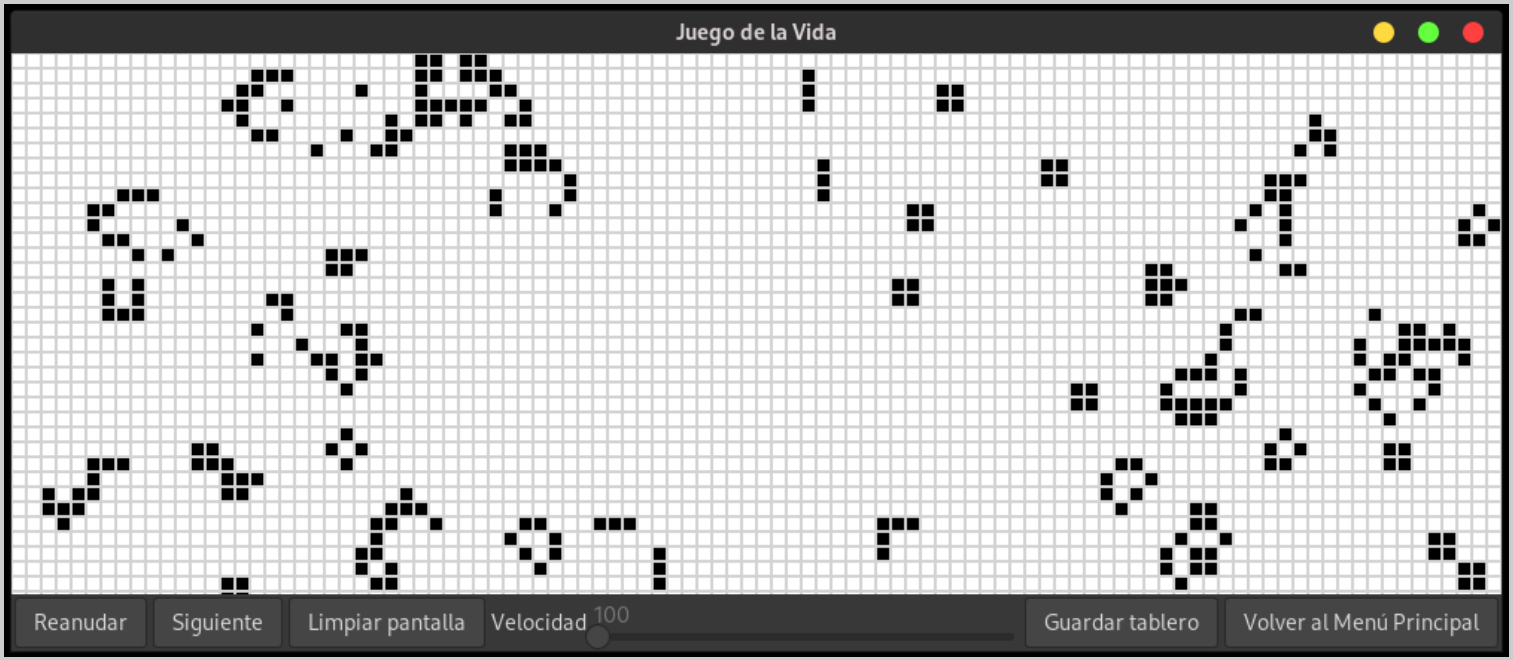


Figure: Pantalla del juego

Módulo: game_logic.rkt - Implementación del *canvas*

```
;; Canvas
(define canvas
  (new (class canvas%
         ; Importar el método para trabajar con el dibujo
         (inherit get-dc)

         ;; Manejo de eventos del canvas
         (define/override (on-event event)
           (when (equal? (send event get-event-type) 'left-down)
             (let* (; Variables de let
                    (mouse-x (send event get-x))
                    (mouse-y (send event get-y))
                    (cell-x (quotient mouse-x cell-size))
                    (cell-y (quotient mouse-y cell-size))
                    (index (coords->index cell-x cell-y width))
                    )
               ; Cuerpo de let
               (when (and (>= cell-x 0) (< cell-x width)
                          (>= cell-y 0) (< cell-y height))
                 ;; Actualiza el estado de la celda seleccionada
                 (set! cells (list-set cells index (if (= (list-ref cells index) 1) 0 1))) ; Alterna entre 1 y 0
                 (send this refresh))))) ; Redibuja el canvas

         ; Llama al constructor de la clase `canvas%`
         (super-new))
```

*Truncado

Figure: Implementación del canvas del juego

Módulo: game_logic.rkt - Timer

Para poder modificar la velocidad una vez creada la pantalla, se crea un nuevo *timer* al cambiar dicha velocidad.

```
;; Función para reiniciar el temporizador
(define (update-timer new-interval)
  (when timer
    (send timer stop))
  (set! timer
        (new timer%
             [interval new-interval]
             [notify-callback
              (lambda ()
                (unless paused?
                  (set! cells (update-fn cells))
                  (send canvas refresh)))])))

;; Iniciar el temporizador
(update-timer delay)
```

Figure: Implementación del timer

Resultados

Resultados

El proyecto logra una implementación básica del juego de la vida que permita además:

- **Modificar los parámetros** de juego como el tamaño del tablero, las reglas, la velocidad, etc.
- Generar **tableros aleatorios** con distinto nivel de dispersión.
- **Importar y exportar tableros**. Incluyendo tableros de ejemplo con configuraciones interesantes.

Todo ello accesible mediante una interfaz gráfica clara y versátil.

Demostración

Enlace del proyecto en GitHub:
https://github.com/naibu3/juego-de-la-vida

Conclusiones

Conclusiones

Con este proyecto, ha mejorado la destreza programando, sino que:

- Se ha profundizado en la creación de un proyecto en Scheme.
- Se ha aprendido el manejo de librerías gráficas y módulos externos.
- Ha quedado constancia de las ventajas de la programación declarativa y de las virtudes (y desventajas) del lenguaje.

Gracias por vuestra atención

References

1. *Ejemplos de Scheme en la plataforma de Moodle.* Recursos disponibles en Moodle para los estudiantes. 2024.
2. García, G. L. *Conway's Game of Life.* Trabajo Fin de Grado, Universidad Politécnica de Madrid, Escuela Técnica Superior de Ingenieros Informáticos. Tutor: Jonathan Sánchez Hernández. June 2022. `https://github.com/glopez42/game-of-life.git`.
3. Gardner, M. Mathematical Games: The fantastic combinations of John Conway's new solitaire game "Life". *Scientific American* **223,** 120–123 (Oct. 1970).
4. Racket Development Team. *Racket GUI Library Documentation.* Accessed: 2024-06-17. 2024. `https://docs.racket-lang.org/gui/`.
5. *Trabajos realizados por otros estudiantes en cursos anteriores.* Referencias a proyectos académicos previos. 2024.
6. Wikipedia contributors. *Juego de la vida — Wikipedia, la enciclopedia*